

Intelligent transport systems — Navigation systems — Application programming interface (API)

ICS 03.220.01; 35.240.60

National foreword

This British Standard is the UK implementation of ISO 17267:2009.

The UK participation in its preparation was entrusted to Technical Committee EPL/278, Road transport informatics.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2010
© BSI 2010

ISBN 978 0 580 64723 9

Amendments/corrigenda issued since publication

Date	Comments

INTERNATIONAL STANDARD

BS ISO 17267:2009

ISO
17267

First edition
2009-11-15

Intelligent transport systems — Navigation systems — Application programming interface (API)

*Systèmes intelligents de transport — Systèmes de navigation —
Interface de programmation (API)*



Reference number
ISO 17267:2009(E)
Documents
Document Sharing Hub
© ISO 2009

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO 2009

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword	iv
Introduction.....	v
1 Scope	1
2 Terms and definitions	1
3 Abbreviated terms	7
4 Architecture of the API.....	8
4.1 General	8
4.2 Paradigm	8
4.3 Minimum low level platform interface	8
4.4 Forward compatibility	8
4.5 Error handling.....	8
4.6 Memory allocation	9
4.7 Prioritization and cancellation	9
4.8 Byte ordering	9
4.9 Generic data types	9
4.10 Handling of large result sets	10
4.10.1 Background.....	10
4.10.2 Requirements.....	11
4.10.3 Object-oriented example.....	11
4.11 Multimedia issues.....	13
4.12 Location of application software, DAL and data.....	13
4.12.1 Application software	13
4.12.2 Data access library.....	14
4.12.3 Data	14
4.12.4 Conclusions	15
4.13 Base and extended APIs.....	21
4.13.1 Terms	21
4.13.2 Description	21
5 Functional specification of the API	22
5.1 Introduction and level of API.....	22
5.1.1 General	22
5.1.2 Functional definition of the API level	23
5.1.3 ISO-API level policy	24
5.2 Specification convention	25
5.2.1 General	25
5.2.2 Naming conventions	25
5.2.3 Hungarian notation convention	26
5.3 Application categories	30
5.3.1 General	30
5.3.2 Global module specification	31
5.3.3 Definitions common to all functional categories	31
5.3.4 Route planning	70
5.3.5 Route guidance.....	87
5.3.6 Positioning	92
5.3.7 Map display	93
5.3.8 Address location	100
5.3.9 Services/POIs.....	109
5.3.10 Utility functions	114
Annex A (normative) Condition policy	121
Annex B (normative) Attribute types	129
Bibliography.....	139

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 17267 was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

Introduction

The impetus for this International Standard was the recognition by the intelligent transport systems (ITS) industry of the need for standardization with respect to data access for map databases used by navigation applications. As the vehicle navigation industry has grown, so has incompatibility between navigation systems and map databases. Both a standardized physical storage format (PSF) and a standardized navigation application programming interface (API) can facilitate the interoperability between navigation systems and map databases.

The purpose of this International Standard is to define and structure the model for data access for Vehicle Navigation and Traveller Information Systems. This International Standard is not restricted to physical media and will be independent of any underlying physical storage format. While this API is primarily targeted at self-contained in-vehicle systems, it is expected to be usable by other applications that use map data results in essentially the same way. For example, it may be usable by client/server or distributed navigation systems and location-based services without further specialization.

This International Standard is the Application programming interface (API) specification. It represents the comprehensive specification of the API standard for navigation applications. This International Standard builds upon, and is consistent with, the other International Standards developed by ISO/TC 204/WG 3:

- ISO 14825, *Intelligent transport systems — Geographic Data Files (GDF) — Overall data specification*;
- ISO 17572 (all parts), *Intelligent transport systems (ITS) — Location referencing for geographic databases*.

Intelligent transport systems — Navigation systems — Application programming interface (API)

1 Scope

This International Standard specifies an application programming interface (API) for navigation systems. It specifies the data that may be retrieved from the map database and defines the interface for access. This International Standard specifies a set of function calls. It also specifies the design of the API and gives examples of its intended use. Furthermore, it gives the criteria to determine whether a data access library is in accordance with this International Standard.

This International Standard is applicable to the following functional categories of navigation applications:

- positioning;
- route planning;
- route guidance;
- map display;
- address location;
- services and point of interest (POI) information access.

2 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1

address location

application category that deals with the task of expressing a real world position in terms of the data representation

NOTE Address location is one of the six application categories supported by the API.

2.2

address type

attribute of road section entity that specifies the type of house number ranges

EXAMPLE Distinction between base address, county address, commercial address, etc., or no address.

2.3

application category

basic sub-function within the set of functionality for vehicle navigation and traveller information system applications

NOTE This International Standard identifies six application categories: positioning; route planning; route guidance; map display; address location; services and POI information access.

2.4 **application programming interface** **API**

standard interface and set of function calls between application software and data access libraries of vehicle navigation systems, in accordance with this International Standard

2.5 **base map**

all transportation elements and all services, including their relationships to transportation elements

2.6 **branded third-party data** **BTPD**

information about services which is supplied by third-party data providers (e.g. tourist or motoring organizations), who may impose proprietary restrictions on the use and presentation of the data

NOTE 1 Access is subject to authorization and licensing.

NOTE 2 BTPD is a subset of third party data (TPD); see 2.54.

2.7 **cartographic feature**

data model entity that represents geometrical information for display purposes

NOTE A cartographic feature has non-explicit topology; it has zero-, one- and two-dimensional types, i.e. Display Point, Polyline, and Polygon.

2.8 **cartographic text**

data model entity that stores name text associated with all or part of a cartographic feature

NOTE Cartographic text is language dependent and may contain a suggested display location, orientation, language code, priority (or importance), suggested scale range, and bounding box.

2.9 **condition**

information related to link(s) composed of condition type, condition modifiers, and condition scope

2.10 **crossroad**

data model entity that represents the single instance of the crossing of two named navigable features

NOTE Crossroad relates to the set of links and nodes which comprise the crossing, and to the crossing of the navigable features to a place.

2.11 **destination node**

node at the end of the link toward which travel takes place

NOTE See also origin node (2.25), "from" node (2.14), "to" node (2.55), source node (2.50), and target node (2.53). When a link is travelled in the direction of topological orientation, the destination node is the "to" node. When it is travelled in the direction opposite topological orientation, the destination node is the "from" node.

2.12 **display point**

zero-dimensional type of cartographic feature

2.13

dummy point

optional entity that represents a position along a link where the link crosses a parcel boundary and does not necessarily coincide with a shape point or node

2.14

“from” node

node at the end of a link away from which the link is topologically oriented

NOTE See also “to” node (2.55), origin node (2.25), destination node (2.11), source node (2.50), and target node (2.54). When a link is travelled in the direction of topological orientation, the “from” node is the origin node. When it is travelled in the direction opposite topological orientation, the “from” node is the destination node.

2.15

geocoding

determination of a link or node based on address information describing and/or naming a location

2.16

intersection

geographic data file (GDF) level 2 representation of a crossing which bounds a road or a ferry as a complex feature composed of one or more GDF level 1 junctions, road elements and enclosed traffic areas

2.17

junction

data model entity that represents a navigable feature which is either a named GDF junction or named GDF intersection, and that relates a named navigable feature to a set of links and nodes and a place

2.18

landmark

point, line, or area feature, possibly associated with a node or link, that can be used to clarify the directions generated to describe a route

NOTE A landmark may not be in the Services, Administrative Areas, or Public Transportation feature themes of a GDF; a facility in which a service is located may be a landmark.

2.19

layer

subset of map data resulting from a subdivision of data of the same coverage area based on contents and which is typically related to one or only a few of the application categories

NOTE This is similar to an ISO-GDF layer.

EXAMPLE Route guidance data may be considered as one layer.

2.20

level

subset of map data resulting from a classification of data of the same semantic content based on the level of detail or density, related to the concept of different map scales

NOTE Level 0 is considered the lowest level (greatest detail); higher levels are numbered level 1, level 2, etc.

EXAMPLE Map display data may be organised into 6 levels representing different zoom scales.

2.21

link

directed topological connection between two nodes, composed of an ordered sequence of one or more segments and represented by an ordered sequence of zero or more shape points (2.48)

2.22

map display

application category that deals with graphical information presentation

NOTE Map display is one of the six application categories supported by the API.

2.23

multilink

ordered aggregation of links which are at the same level, are connected in sequence, and share the same functional classification, form of way, direction of travel, and perhaps additional characteristics

EXAMPLE Each link is contained in exactly one multilink.

2.24

navigable feature name

data model entity that represents the name for the transportation element, including GDF road element, GDF ferry connection, GDF junction, GDF intersection

NOTE Navigable feature name is related to places, crossroads, junctions, and road sections.

2.25

node

data model entity for a topological junction of two or more links or for end-bounding a link

NOTE A node stores the coordinate value of the corresponding GDF junction.

2.26

origin node

node at the end of a link from which travel takes place

NOTE See also destination node (2.11), “from” node (2.14), “to” node (2.55), source node (2.50), and target node (2.54). When a link is travelled in the direction of topological orientation, the origin node is the “from” node. When it is travelled in the direction opposite topological orientation, the origin node is the “to” node.

2.27

parcel

database partitioning unit corresponding to a certain coverage area, associated with one level and containing data of one or more layers

NOTE A parcel contains (at least) all nodes with positions enclosed by or located on the outline of its coverage area plus (parts of) all links attached to these nodes; it can be partitioned so that the amount of data of a parcel may be nearly the same as that of another.

2.28

place

named area which can be used as part of the address location

2.29

place class

attribute of place entity, classifying into highest administrative or geographic division, administrative subdivision, postal, or colloquial (e.g. regions or neighbourhoods)

NOTE Place class can be partially ordered as “place class A is below place class B”. This does not imply strict or complete containment.

2.30

place level

level associated with places of place classification “administrative subdivision”

NOTE Higher/lower level situations are constituted by the occurrence of a parent/child place relationship between places.

2.31

place relationship

bivalent relationship between place entities, constituting the place tree linking parent and child places

EXAMPLE Place A is in place B.

NOTE Place relationship does not imply strict or complete containment. It is attributed as: address significant, official, postal or useful for reverse geocoding.

2.32

point of interest

POI

destination and/or site of interest to travellers, usually non-commercial by nature

2.33

polygon

two-dimensional type of cartographic feature

2.34

polyline

one-dimensional type of cartographic feature

2.35

positioning

application category that deals with the determination of vehicle location and map matching

NOTE Positioning is one of the six application categories supported by the API.

2.36

postal code

data model entity for a government-designated code used to specify regions for addressing

NOTE Postal code is related to link (2.21), navigable feature name (2.23), place (2.27), and POI (2.31).

2.37

rectangle

unit of geographic space defined by two parallels of min./max. latitude and by two meridians of min./max. longitude and that represents the coverage area of the map data enclosed by or located on its outline

2.38

reverse geocoding

determination of the address description of a link or node, i.e. determination of an upwards path across the place tree

2.39

road

GDF level 2 feature composed of one, many or no road elements and joining two intersections, serving as the smallest independent unit of a road network at GDF level 2

2.40

road element side

RES

basic component of the road section entity that represents left or right side of a link and corresponds to one or more unique combinations of a navigable feature and a house number range

2.41

road section

data model entity that represents the house number ranges of both sides of a street and that carries a navigable feature name

NOTE Road section corresponds to a link (ID).

2.42

route guidance

application category that deals with the generation of graphical, textual, and/or audio instructions for following a planned route

NOTE Route guidance is one of the six application categories supported by the API.

2.43

route planning

application category that deals with the determination of routes between specified points

NOTE Route planning is one of the six application categories supported by the API.

2.44

segment

straight section of a link connecting two successive shape points, or a shape point and a node, or two nodes where a link does not contain shape points

2.45

service

data model entity for a commercial activity of interest to travellers as a destination and/or orientation that is associated with road element(s) by which it can be accessed and further described by attributes including (at least) name and type

NOTE A service may be associated with other services by parent/child relationships (many to many). Service is used synonymously with POI within the logical data model.

2.46

service attribute

item of descriptive information relating to a service

2.47

services and POI information access

application category that deals with the provision of POI information to the navigation application

NOTE Services and POI information access is one of the six application categories supported by the API.

2.48

shape point

position along a link used to more accurately represent its geometric course, bounded by exactly two segments

2.49

signpost

data model entity for a directional sign that represents a logical relationship between signpost information and two associated links

NOTE The first link (mandatory) represents the road element along which the signpost is located. The second link (optional) is the first road element which directs exclusively to the destination indicated on the signpost. The position of the signpost along the link and the link direction the signpost is facing is also stored.

2.50

source node

node at the end of a link from which exploration takes place for route calculation

NOTE See also target node (2.53), origin node (2.25), destination node (2.11), "from" node (2.14), and "to" node (2.55). When forward exploration is taking place from the origin of the route, the source node of a link is its origin node. When reverse exploration is taking place from the destination of the route, the source node of a link is its destination node.

2.51

super link

aggregation of linearly connected regular links present in the lowest level as a simplified representation of the road network in higher levels

2.52

symbol

data model entity that represents an icon associated with a cartographic feature

2.53

target node

node at the end of a link towards which exploration takes place for route calculation

NOTE See also source node (2.50), origin node (2.25), destination node (2.11), “from” node (2.14), and “to” node (2.56). When forward exploration is taking place from the origin of the route, the target node of a link is its destination node. When reverse exploration is taking place from the destination of the route, the target node of a link is its origin node.

2.54

third party data

TPD

information about services, which is supplied by third party data providers (e.g. tourist or motoring organizations), typically with a rich content of descriptive data

2.55

“to” node

node at the end of a link towards which the link is topologically oriented

NOTE See also “from” node (2.14), origin node (2.25), destination node (2.11), source node (2.50), and target node (2.54). When a link is travelled in the direction of topological orientation, the “to” node is the destination node. When it is travelled in the direction opposite topological orientation, the “to” node is the origin node.

2.56

traffic location

data model entity that contains an external reference (e.g. VICS or RDS-TMC) and is linked to either place or transportation entities

2.57

transportation element

feature from the Roads and Ferries feature theme of a GDF

3 Abbreviated terms

ANSI	American National Standards Institute
CPU	Central Processing Unit
DAL	Data Access Library
DBID	Database ID
DST	Daylight Savings Time
EEPROM	Electrically Erasable and Programmable Read-Only Memory
GDF	Geographic Data File
GMT	Greenwich Mean Time

HOV	High Occupancy Vehicle
HTML	HyperText Markup Language
IDL	Interface Definition Language
MIME	Multipurpose Internet Mail Extensions
OMG	Open Management Group
OS	Operating System
PSF	Physical Storage Format
RDS-TMC	Radio Data System-Traffic Message Channel
VICS	Vehicle Information and Communication System

4 Architecture of the API

4.1 General

Subclauses 4.2 to 4.13 specify the architecture requirements for the design of the ISO-API. Implementation details are not specified in this International Standard.

4.2 Paradigm

The ISO-API shall be specified in an object-oriented way.

4.3 Minimum low level platform interface

It is not necessary to define a minimum low level platform interface as long as a system-independent data access library is not feasible.

4.4 Forward compatibility

The ISO-API shall support forward compatibility in such a way that

- earlier versions of application software can use DALs corresponding to later ISO-API versions, and
- earlier versions of DAL can use data in later PSFs.

The mechanism for forward compatibility shall be entirely hidden from the API.

4.5 Error handling

The application software shall be responsible for handling errors; more specifically, the DAL gives notification of errors while the application software reacts on them.

The ISO-API shall specify a list of error conditions for each function call. The system designer can decide which error mechanism to use for the implementation.

4.6 Memory allocation

The lifetime of the objects and structures shall be controlled by the application software. Memory for objects and structures used internally by the DAL is managed by the DAL.

4.7 Prioritization and cancellation

The ISO-API shall support prioritization and cancellation to control input/output (I/O) operations, perform intelligent caching, etc. This functionality shall be supported by the ISO-API in the following way: Each class (object-orientation!) shall have two member functions "setPriority()" and "getPriority()". Due to this, the application software is able to assign, query and change the priority for an instance of this class. If the application software does not want to use this mechanism, it can assign the same value for each instance. There is no mandated behaviour of the DAL based on the application-specified priorities. The behaviour of the DAL is defined by the DAL supplier.

4.8 Byte ordering

Depending on the hardware (and operating system) of a navigation system, the data values will be physically used in one of two ways.

- Little endian: The least significant byte (LSB) of a value will be stored physically first, e.g. in systems with Intel CPUs.
- Big endian: The most significant byte (MSB) of a value will be stored physically first, e.g. in systems with Motorola 68k CPUs.

Some CPUs are bi-endian, such as R3000 ff, PowerPC. In such a case, the byte order is defined by the operating system.

The following data values are concerned:

- integer values (16, 32, 64 bit);
- floating point values (32, 64, 128 bit);
- characters (16 bit, multibyte).

An ISO-API compliant DAL shall return the data values in the byte order being used by the CPU/OS of the navigation system where the DAL is running. This shall happen regardless of whether a PSF is stored in little endian or big endian byte order. Therefore, a conversion inside the DAL may be necessary. The performance loss shall be minimized by doing the conversion on the lowest level.

4.9 Generic data types

Because the target systems are different it is necessary to define generic system-independent data types to be used for the ISO-API. They are translated by the DAL into the data types of the target system. The generic data types listed in Table 1 shall be defined for the ISO-API.

Table 1 — Generic data types

Meta type	IDL type	Size/Format	Description
MapByte ^a	octet	8-bit two's complement	Byte-length integer
MapShort ^a	short	16-bit two's complement	Short integer
MapLong ^a	long	32-bit two's complement	Integer
MapLongLong ^b	longlong	64-bit two's complement	Long integer
MapFloat ^b	float	32-bit IEEE 754	Single-precision floating point
MapDouble ^b	double	64-bit IEEE 754	Double-precision floating point
MapChar ^c	wchar char	16-bit or less character 8-bit character	Single character Single character
MapBoolean	Boolean	8-bit; 0x00 is false, any other is true	Boolean value: true or false
MapString	string	16-bit or less characters; max. length 65 535	Zero-terminated string
^a Unsigned and signed. ^b These types may not be supported natively on some systems. Their inclusion here does not necessarily mean that they have to be used. ^c The size of the characters to be used depends on the definitions in the PSF. Variable length character sets are not necessarily considered yet.			

The column entitled “IDL type” in Table 1 establishes the reference between the well-defined and stable meta type and the implementation in IDL.

4.10 Handling of large result sets

4.10.1 Background

For many API calls, the amount of data that will be returned will not be easy to predict in advance. For example, a call to retrieve all road elements within a bounding box may return a very large amount of data if the bounding box falls in a dense area.

On the other hand, DALs will be implemented on a wide variety of platforms. If memory for results is allocated dynamically, returning very large results may pose problems for memory allocation on some platforms.

A flexible solution is required, so that the result can be returned in pieces if necessary. The proposal below specifies such a solution. The general approach is as follows: One function is called to prepare for the return of a possibly large result set. A second function is called as many times as necessary to return the entire result set. A third function is called, perhaps before the entire result set has been returned, to “close” the operation.

The first and second functions return an indication of the size of the result set to be expected, so that applications which are capable of allocating large amounts of memory dynamically can allocate enough memory to receive the whole result set at once, thereby improving efficiency. However, there are cases in which it is computationally expensive to compute the exact size of the result set in advance; if the caller never asks for the whole result set, such computation is wasted. Therefore, the proposal allows for an overestimate, rather than an exact size, to be returned.

The following architectural requirement in 4.10.2 is therefore specified.

4.10.2 Requirements

Calling sequences for functions that may return results of unpredictable size shall be structured as follows:

- a) Functions to return such result sets shall come in sets of three:
 - 1) The first function (the "opening" function) describes the request, initializing a data structure (herein called a "handle"), which is used to store the state information for the request.
 - 2) The second function (the "fetching" function) is called to return successive parts of the result. However, this function need not be called enough times to return the entire result; indeed, it need not be called at all.
 - 3) The third function (the "closing" function) is called to notify the library that the application is finished with the request. The use of handles allows more than one request to be in process at the same time. The library may place a reasonable upper limit on the number of handles that are active at one time. (The question of whether the memory for the handles is allocated by the library or by the application must be decided but is not addressed here.)
- b) The opening function shall return a value indicating the number of bytes expected to be necessary to hold the result. Also, each call to the fetching function shall return a value indicating the number of bytes expected to be necessary to hold the remainder of the result. In order to allow more efficient implementation, these values may be overestimates, rather than exactly correct values. A special value is also reserved to indicate that no good (over)estimate can be provided. (Using the maximum positive integer as this special value is recommended but not required for conformance with this International Standard.)
- c) One argument to the fetching function is a pointer to the buffer in which the result is to be returned. Another argument is the size of the buffer in bytes. This buffer (if necessary) shall be allocated and de-allocated by the calling application, not by the functions described here. Also, the address and size of the buffer need not be the same from call to call.
- d) The fetching function shall return the number of objects returned with each call. In order to accommodate objects of variable size, the function shall also return the number of bytes returned.
- e) The opening function shall also return an indication of whether the result will be non-empty, and the fetching function shall also return an indication of whether it has returned the entire result. (This can be derived from whether the estimates of the number of bytes required for the result or for the remainder of the result, respectively, is zero. However, that approach may be considered needlessly obscure, in which case a separate return value, for example the function's return value, should be used.) However, for purposes of efficiency, these indications must be allowed to give false positives, that is, to indicate that return values exist when in fact none do; this allows the determination of whether the values actually exist to be postponed to the next call to the fetching function.
- f) The functions may defer disk reads, record unpacking, and similar operations until they are needed. Since it is expected that, in many instances, the fetching function will be called to return only part of the full result, this may result in substantial gains in efficiency.

4.10.3 Object-oriented example

The following example describes an object-oriented implementation of the same functionality. A class instead of a handle is used to store the state information for the query, and the constructor and destructor methods for the class serve as the "opening" and "closing" functions, respectively. The user-defined type "object_t" has been replaced with the class MapObject. Exceptions are used instead of negative return values to indicate failures. All other features of the design, including the allocation of memory by the application, are similar to the procedural example above.

```
//
// query interface
//
class MapQuery {
public:
    MapQuery([parameters specifying objects to return]) throw
    (open_failed);
    ~MapQuery(void) throw (close_failed);
    bool fetch(void) throw (fetch_failed);
    void setBuf(MapObject* bp);
    void setBufSize(size_t s);
    size_t getResultSize(void);
    size_t getResultObjects(void);
    size_t getRemainderSize(void);
};

//
// sample application
//

void main()
{
    MapObject* buf;
    size_t numObjects;

try
{
    // query map
    MapQuery query([parameters specifying objects to return]);

    // obtain prospective number of matching objects
    numObjects = query.getResultObjects();

    // application allocates memory to hold query results
    buf = new MapObject[numObjects];

    // application tells object location of allocated memory
    query.setBuf(buf);
    query.setBufSize(size of(MapObject) * numObjects);

    // process query
    while (query.fetch())
    {
        for (int i=0; i<query.getResultObjects(); ++i)
        {
            buf[i].handleObject();
        }
    }

    // handle failures in MapQuery methods
    catch (MapException e)
    {
        cerr << e.what() << endl;
    }
    // application deallocates memory
    delete [] buf;
}
}
```

4.11 Multimedia issues

In addition to the map data, the database may contain multimedia objects, e.g. an image of a POI. It shall be possible to retrieve such information via the API.

The following types of multimedia object shall be supported:

- text files;
- HTML files;
- sound;
- still images;
- motion images.

The set of multimedia types that are supported by the API is not fixed, i.e. it shall be extendible.

The API shall be able to identify the type of a multimedia object so that the application software is able to associate it with the appropriate decoder/presentation subsystem. Therefore, a content type specifier shall be supported in the API for each multimedia object. The content type specifier shall be compliant with the MIME (Multipurpose Internet Mail Extensions).

Furthermore, the API shall allow the use of stream I/O for multimedia objects on platforms which support it.

Multimedia decoders and presentation subsystems are outside of the scope of this International Standard.

4.12 Location of application software, DAL and data

4.12.1 Application software

The application software implements

- the complete user interface of the navigation software (i.e. map display, user interaction elements), and
- the system-dependent parts (i.e. O/S level memory management, input/output).

There are four possibilities for the location of the application software within a certain navigation system:

- 1) Stored permanently in the system (i.e. EEPROM):
 - it can be easily accessed;
 - application updates are non-trivial, especially for the large number of systems on the market.
- 2) Stored together with the data on removable media (generally on CD-ROM):
 - requires that the application be pre-loaded into the system before execution;
 - allows easy updating of the application software (and data);
 - not suitable for PSF-compatible systems.

- 3) Stored separately on removable media (i.e. flash EEPROM card):
 - requires that the application be pre-loaded into the system before execution;
 - applications can be updated independently of data.
- 4) Any combination of 1) to 3):
 - combines the disadvantages of the used possibilities.

4.12.2 Data access library

The data access library consists of all functions (presently 6 application categories) that are accessing data. For the DAL location there are the following cases:

- 1) Stored permanently in the system:
 - can be easily accessed;
 - DAL updates are complicated.
- 2) Stored together with the data on removable media:
 - requires that the DAL be pre-loaded into the system;
 - DAL and PSF are always compatible;
 - DAL updates at the same time as the data.
- 3) Stored separately on removable media:
 - requires that the DAL be pre-loaded into the system;
 - DAL updates can be done independently of data.
- 4) Any combination of 1) to 3):
 - combinations of 2) and 3) will raise the flexibility of the whole system.

4.12.3 Data

Data in the context of this International Standard is the summary of all geographical data that are required for certain functional categories. There are the following possible locations for the data:

- 1) Stored permanently in the system:
 - data updates are non-trivial due to the high update rate of the data and the large amount of data.
- 2) Stored on removable data media:
 - data updates are much easier;
 - the approach generally used for navigation systems.
- 3) Combination of 1) and 2):
 - combines the disadvantages of 1 with 2.
- 4) Accessible over the Internet or another network, either wired or wireless.

4.12.4 Conclusions

This International Standard does not place restrictions on the locations of any component. A more flexible solution allows for competition between the systems.

Figures 1 to 3 display examples showing several possibilities for the location of components.

Figure 1 a) shows how two different systems (hardware A with application software A, and hardware B with application software B) can be serviced by a single data medium. It is assumed that each navigation system can interface to

- any industry-wide OPEN PSF (by having a proper DAL built into its application software),
- the ISO-API (by having the ISO-API functions defined in its application software), and
- its own proprietary API-A or its proprietary PSF-A, both of which are outside the scope of this International Standard.

Hardware A (or system A) refers to both the operating system and the computing hardware. Should either change, it would no longer be system A.

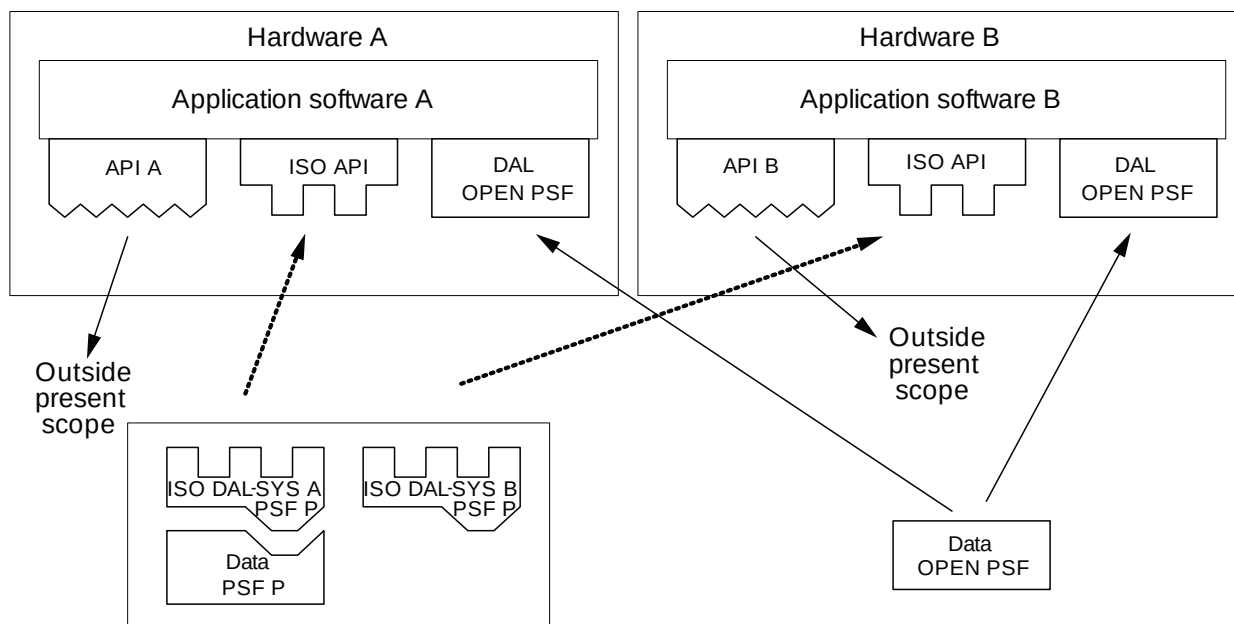
The medium on the right-hand side consists of data only, and it is in an OPEN PSF format. Both navigation systems can read an OPEN PSF via their built-in OPEN PSF DALs.

The medium on the left-hand side delivers its data in some proprietary PSF-P. The same medium has a DAL, which reads the proprietary PSF-P and returns that data in a format in accordance with this International Standard. The DAL is either in Java or in compiled binary code for hardware A or hardware B. In this scenario, the DALs are supplied on the medium along side the data. In Figures 2 and 3, the DALs are supplied independently of the data media. Given that the DAL software is much smaller in size than the actual data, there can be as many versions of it as there are known navigation systems. While this method is not forward compatible, neither is the quality and content of the data. Should new navigation systems come into existence, compiled versions of the DAL for PSF-P can be delivered the next time the data is updated and a new set of media is released.

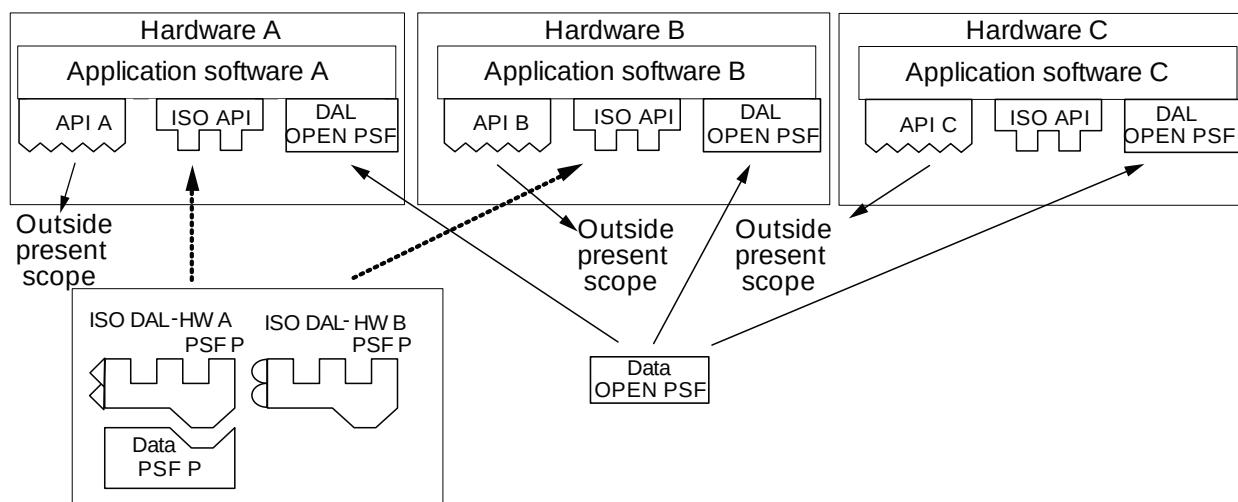
In Figures 1 b) and 1 c) the specific situation of forward compatibility is explored. Upon release of the medium with the set of DALs for all known systems at that time, the medium is fully interoperable. However, if the DAL is not a Java DAL (or a new system C shows up which cannot support Java), then the only means of interoperability with the future system C is via an OPEN PSF.

Figure 1 b) shows the situation of forward compatibility. A new system (system C) is introduced into the market. The medium does not contain the ISO-API compiled for system C. There are only two ways, given that medium, to be able to operate on system C: either the Java DAL is on the medium (and system C can make use of it), or an OPEN PSF is on the medium (which would have to be the case in order to be able to claim that medium compliant with a particular OPEN PSF). In this particular situation the ISO-API would not be (forward) compatible with the new system C. Figure 1 c) shows one possible remedy.

In Figure 1 c) a remedy to the failure of total forward compatibility of the ISO-API is proposed. Via collaboration between data supplier P (presumably the owner or licensor of PSF format P), the ISO-DAL creator (data supplier P or related to data supplier P), and the new system maker C, a supplemental DAL is made available, either with or without the data published in PSF P. If map data is supplied, potentially it is a newer and better version than that supplied originally.

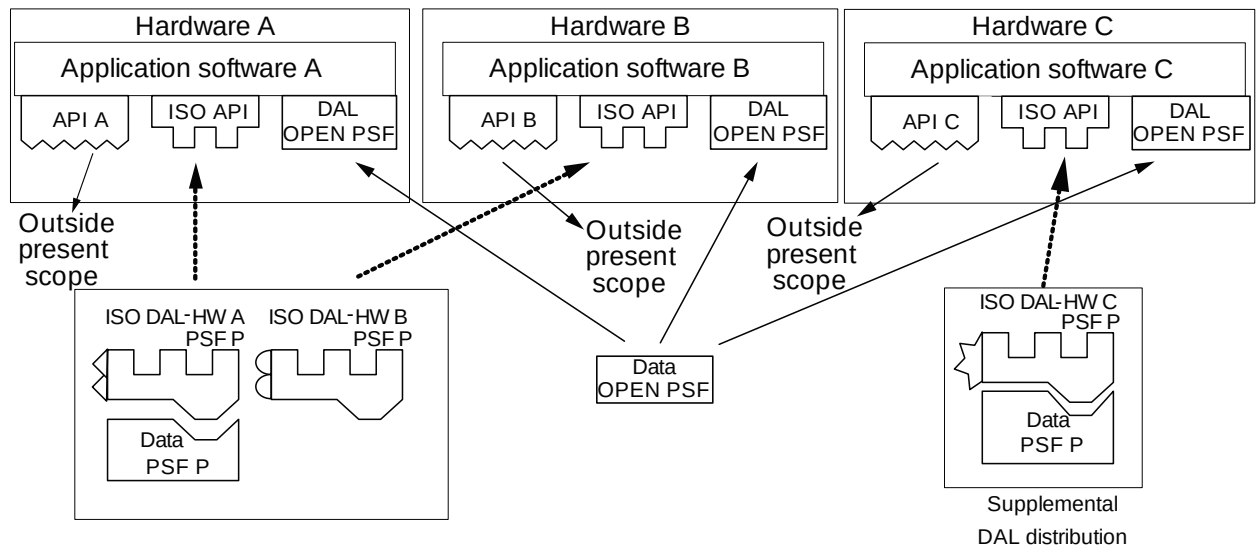


a) As supplied



b) A new system C is introduced into the market

Figure 1 — (DAL on the medium)



c) A supplement goes out to address the new system

Figure 1 (continued)

In Figure 2 a) the DALs are supplied independently of the data. Each system vendor supplies the DALs for the PSFs that it chooses to support using an access mechanism in accordance with this International Standard. The DALs are pre-loaded and therefore part of the application software by the time the data arrives. By having the data delivered in an ISO-API access mechanism, the application software can access the data in a uniform way, regardless of the delivery mechanism.

Similarly to the case of the DAL on medium, Figure 2 b) explores the situation of forward compatibility upon introduction of a new system C into the market. In this case forward compatibility is sustained.

In this scenario, the data on the medium is independent of the DAL which reads it, so, similarly to system providers A and B, it is the responsibility of the new system provider C to provide DALs for its own system which can read PSFs Q and R, and any other ones which are to be served under the ISO-API. The advantage of this scenario is that there is no market lag between the introduction of new system C and the availability of DALs for ISO-APIs that can read all the data media available in the market place. The disadvantage of this scenario is that it is necessary to expose the PSFs Q and R (and any other ones) to system vendor C.

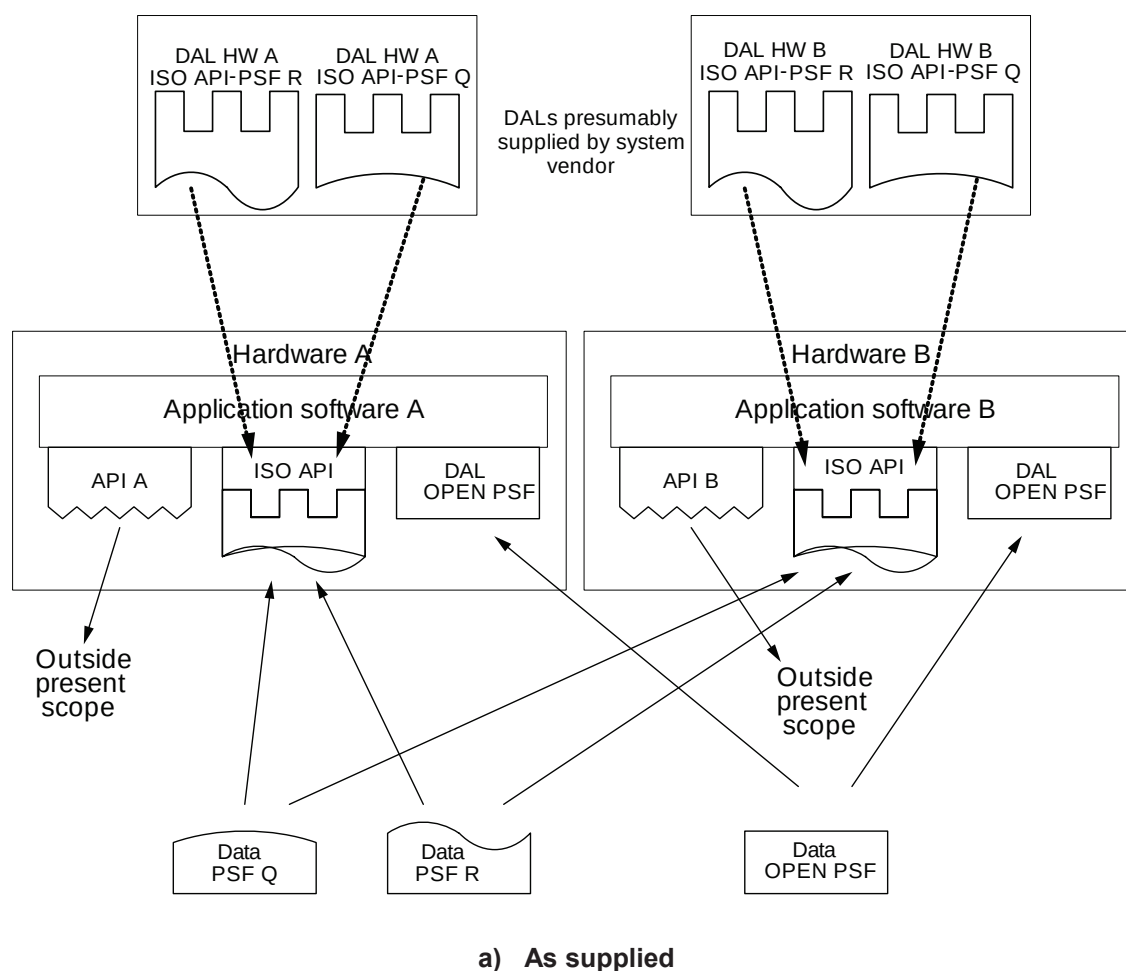
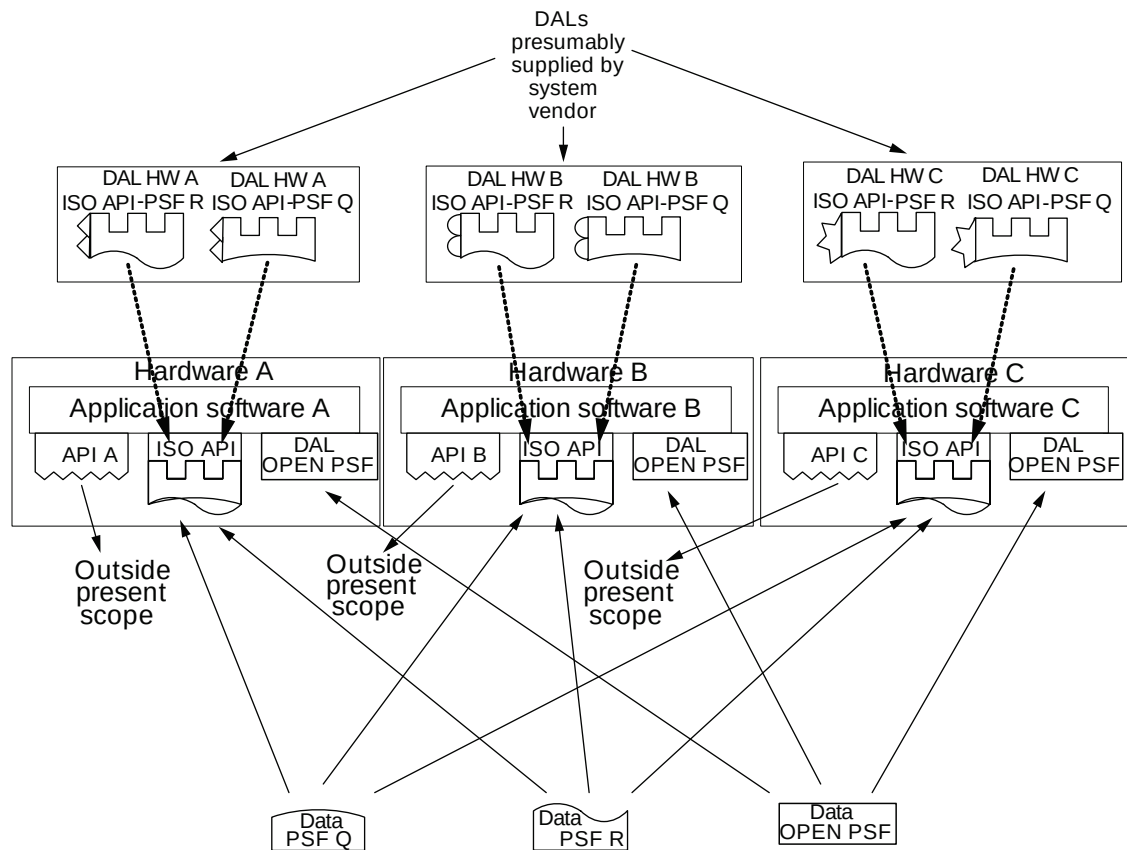


Figure 2 — DAL not on medium, supplied by system maker



b) A new system C is introduced into the market

Figure 2 (continued)

In Figure 3 a) the situation is similar to the situation of Figure 2 a), except for one aspect: The DALs are supplied by a data supplier of a proprietary PSF, rather than by the system vendors themselves. Each PSF supplier implements the DAL for any navigation system they wish to support. Figure 3 b) shows what happens when a new system C is introduced.

In this scenario, the data on the medium is, also, independent of the DAL which reads it, except here it is the responsibility of the data (or PSF) providers to provide for system C, as they did for systems A and B, DALs for their own PSF which would deploy also on system C in a manner that does not conflict with this International Standard. The disadvantage of this scenario is that there is potentially a market lag between the introduction of new system C and the availability of DALs for APIs that comply with this International Standard which can read all the data media available in the market place. Even if some data providers release quick updates for system C, others might lag and leave system C in a somewhat vulnerable position, as its interoperability to all data on the market would be initially only partial. The advantage of this scenario is that no PSFs need be exposed to system maker C.

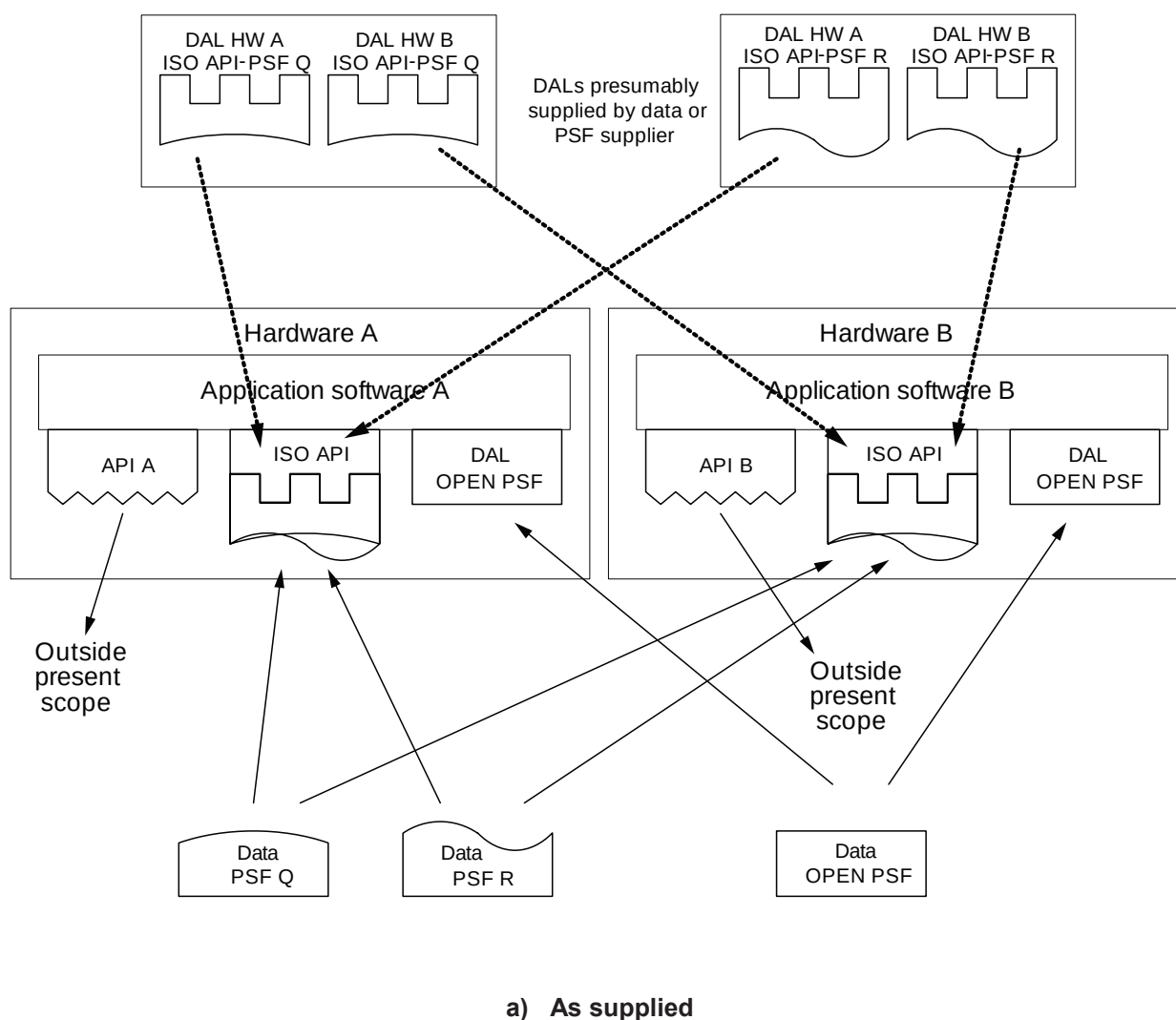
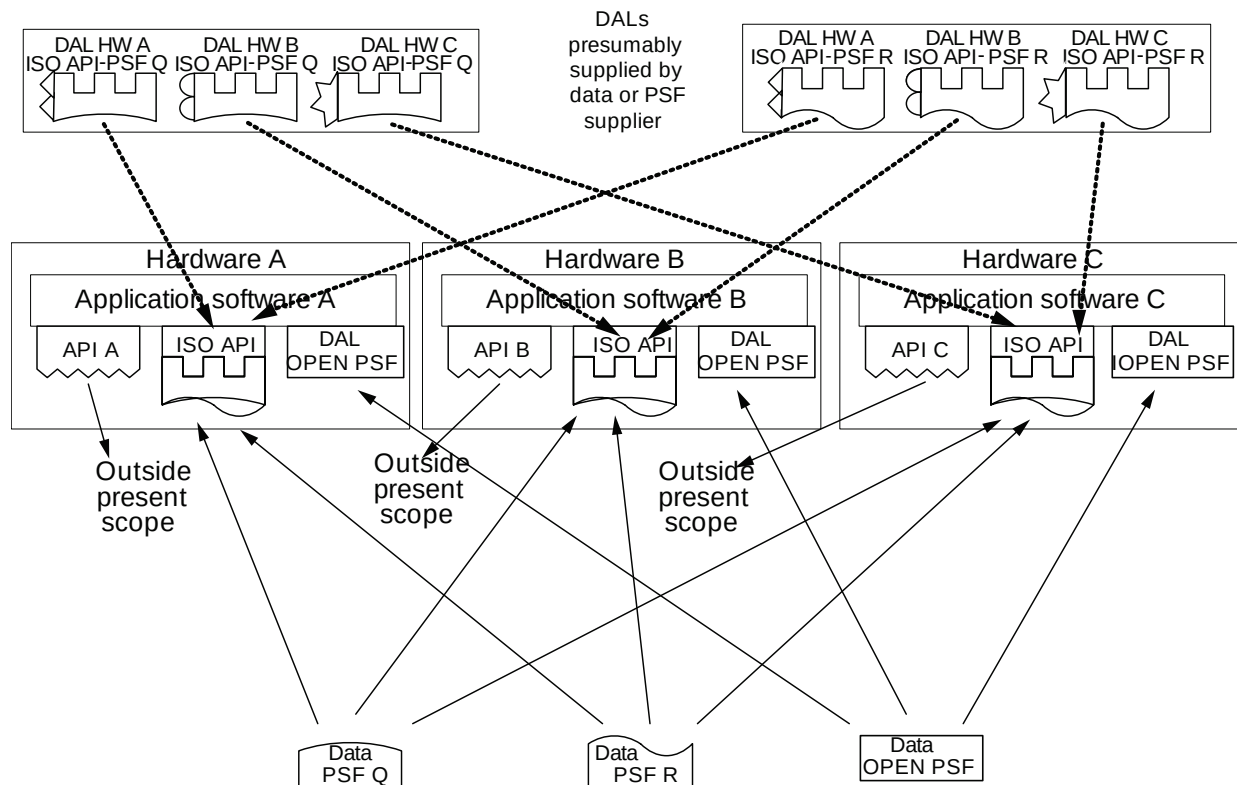


Figure 3 — DAL not on medium, supplied by PSF provider



b) A new system C is introduced into the market

Figure 3 (continued)

4.13 Base and extended APIs

4.13.1 Terms

A Base API contains all *required* functions, and no *extending* functions. Functions are classified as *required* or *extending* according to Clause 5.

An Extended API contains all Base API functions plus all *extending* functions. Only functions explicitly tagged as *extending* in Clause 5 are considered *extending*.

4.13.2 Description

The application shall be able easily to determine whether a DAL supports a base or extended API.

The underlying rule for *extending* functions is that every parcel exposing function is an *extending* function. A function that takes either a parcel or a parcel ID as input, or gives a parcel or a parcel ID as output, is considered a “parcel exposing function”.

5 Functional specification of the API

5.1 Introduction and level of API

5.1.1 General

The purpose of this subclause is to establish a framework and an understanding upon which to base the API level in this International Standard.

It was determined that ISO-API level should be defined within the combination layer and the instruction layer levels. To define the API level more clearly, the upper and lower API levels should also be defined as follows. Definition of API levels is shown in Figure 4 below. In this definition, API-5 and API-6 are internal API levels of the application software.

The decision to recommend the ISO-API layer at mostly the combination layer is based on the following criteria:

- a) A higher level API lends itself to a smaller number of more complex functions and therefore to more rapid development.
- b) A sufficiently high API allows the location of the actual map database to be independent of the application's ability to access the content, be it on-board or off-board.
- c) Where necessary, lower level extensions are provisioned for small granularity access (MapDisplay and RoutePlanning).

The problem with the placement of the ISO-API at too low a level (API level 2 and below) is that there is no benefit to an extremely low-level common interface. At the very least, the ISO-API should isolate the application software from the PSF and media, and in addition, should not depend upon a given hardware configuration.

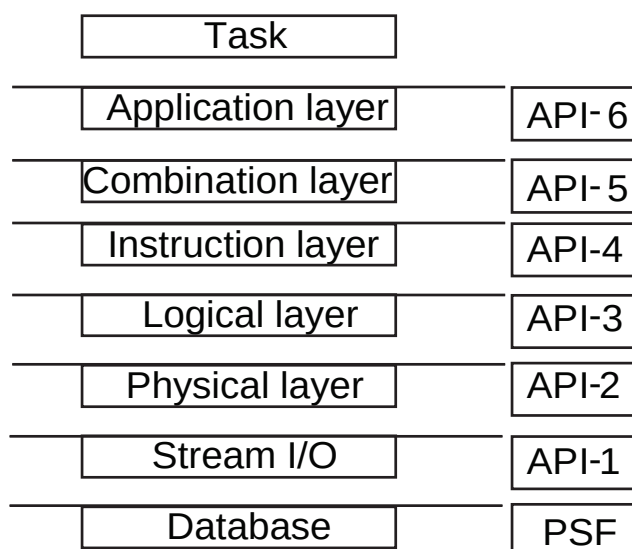


Figure 4 — API levels

5.1.2 Functional definition of the API level

Name of layer	Function of layer
API-6 Application layer	This layer is responsible for high-level system functionality. Examples: Route Planning Vehicle Positioning Route Guidance Map Display
API-5 Combination layer	API-6 layer functions are insulated from lower layers by the API-5 layer. API-5 shades the origin and method of data retrieval. The data returned from the functions in this layer consist of merged lower-level data, and they are returned transparently to the calling function. Examples: Combine data of different types. Get the position of the vehicle on a given link. Return landmark icon codes around a given intersection node. Project map display data for a geodetic coordinate system into the display coordinate system.
API-4 Instruction layer (High logical layer)	API-4 hides data division and data layout from higher layers. Examples: Process data of similar types with multiple access to the physical storage. Process large (memory intensive) data sets. Process data that require multiple searches on the physical media. In such instances, the functions of this particular layer will call the lower level functions multiple times.
API-3 Logical layer (Low logical layer)	API-3 insulates higher layers from the PSF, such as the physical layout of the data structures, like bit fields or word alignment. Functions from this layer convert API-2 PSF into logical structures. Complex operations are done at higher layers. Examples: Return the bounding box of the entire data set. Convert PSF-bytes to structures, including possible byte reordering. Optional data decompression. Convert positions of points from internal representation to latitude/longitude-based values.
API-2 Physical layer	API-2 insulates higher layers from absolute sector addresses. Examples: Read data blocks starting from the specified sector address, up to a given data length. Differentiate according to the file system used.
API-1 Stream I/O layer	Open, read, seek, close. Example: ANSI C file operating functions

5.1.3 ISO-API level policy

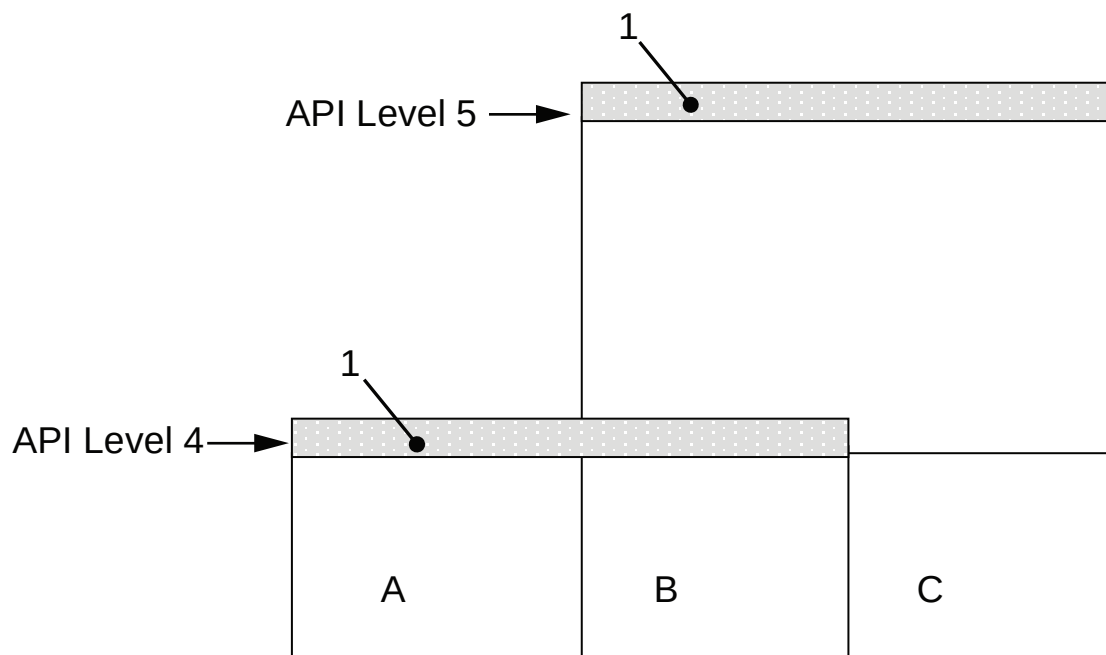
The ISO-API functions can be found either at level 5 or at level 4 (see Figure 5).

There are three categories of functions at level 4:

- functions which are visible through the ISO-API but are not used by any ISO-API level 5 functions (category A);
- functions which are visible through the ISO-API and are used by ISO-API level 5 functions (category B);
- functions which are used by ISO-API level 5 functions but are not visible through the ISO-API (category C).

Each function will be placed at the level and category deemed most appropriate.

Any function which reveals any physical detail of the PSF shall not be visible through the API interface. That is, it may only be of category (C).



Key

- 1 API interface
- A level 4 functions which are not used by any level 5 functions
- B visible level 4 functions which are used by level 5 functions
- C hidden level 4 functions which are used by level 5 functions

Figure 5 — ISO-API level policy

5.2 Specification convention

5.2.1 General

The functional specification of the API is described in the specification for IDL, ISO/IEC 14750. IDL is a system- and implementation-independent description language for software interfaces. Constructs described in this language can be easily implemented by several programming languages, e.g. Java, C++, C.

NOTE This International Standard defines functions and supporting structures, error codes and constants. The structures have been ordered to achieve IDL conformance. The order is therefore not alphabetical.

5.2.2 Naming conventions

5.2.2.1 Additional conventions for the use of IDL to describe the API are specified in 5.2.2.2 to 5.2.2.14.

5.2.2.2 The names of

- classes,
- data types,
- exceptions,
- constants

shall have the prefix "Map".

5.2.2.3 The format of a name is dependent on the type of language element and can be described as follows:

- <ISO-API class name> ::= Map<any class name>
- <ISO-API data type name> ::= Map<any data type name>Type
- <ISO-API exception name> ::= Map<any exception name>Exception
- <ISO-API constant name> ::= Map<any constant name>Const

EXAMPLE class MapRoadElement{...}

5.2.2.4 The name of a member function of a class shall have a prefix and shall start with a capitalized character, e.g. MapGetNode(...).

5.2.2.5 A class member (instance variable) shall start with a capitalized character.

5.2.2.6 Global variables shall be avoided. If really needed they shall be named in the same manner as instances.

5.2.2.7 The single enumerators of an enumeration type shall have a three-digit prefix identifying the enumeration type they belong to.

EXAMPLE

// IDL

typedef enum {AngStraight, AngRight, ... } MapAngleType;

5.2.2.8 In names which consist of more than one word, the words shall be written together and each word that follows the prefix shall begin with an uppercase letter.

5.2.2.9 Whenever possible, entire words or syllables shall be preferred instead of abbreviations.

5.2.2.10 For data elements representing any kind of collection, plurals shall be used rather than inventing new names.

5.2.2.11 Member function names shall be created by using the order Verb/Object, e.g. GetNodeList().

5.2.2.12 The maximum length of a name shall be less than or equal to 32 characters.

5.2.2.13 IDL templates may not be used for the ISO-API because they cannot easily be ported to programming languages such as Java.

5.2.2.14 Multiple inheritance may not be used for the ISO-API because it is not supported by most programming languages.

5.2.3 Hungarian notation convention

Each function argument in this International Standard begins with a prefix which indicates the argument's type. (The practice of using such prefixes is called "Hungarian notation" after the nationality of its inventor, Charles Simonyi.) Implementers and users of this International Standard are encouraged to use these prefixes as well.

Table 2 lists the Hungarian prefixes of the basic data types.

Table 2 — Hungarian prefixes of basic data types

Data type	Prefix
Any	a
Boolean	b
Char	c
Long	l
Octet	o
Short	s
Unsigned long	ul
Unsigned short	us

Table 3 lists, alphabetically by data type, the Hungarian prefixes of the data types defined in this International Standard.

Table 3 — Hungarian prefixes of data types defined in this International Standard

Data type	Prefix
MapAbsoluteDateType	dta
MapAccessVehicleType	av
AddressFieldStatusEnum	afse
MapConditionAndDirectionType	cod
MapConditionAttrType	ca
MapConditionCategoryType	cc
MapConditionLinkAndNodeType	cln
MapConditionModifierEnum	cme
MapConditionModifierType	cm
MapConditionType	cnd
MapCursorInfoType	ci
MapCursorOriginEnum	coe
MapCursorReturnTypeEnum	crte
MapCursorType	cu
MapDateAttrType	da
MapDateTimeType	dt
MapDateType	dt
MapDateTypeEnum	dte
MapDayOfMonthOfYearType	dmy
MapDayOfWeekEnum	dwe
MapDayOfWeekOfMonthOfYearType	dwmy
MapDayOfWeekOfMonthType	dwm
MapDayOfWeekOfYearType	dwy
MapDBIDType	dbid
MapEntityEnum	ee
MapEntityIDType	eid
FeatureTypeEnum	fte
MapFetchDirectionEnum	fde
MapLaneCategoryType	lc
MapLinkAttrType	la
MapLinkCharacteristicsType	lch
MapLinkSubattrType	ls
MapLinkType	lk
MapNodeAttrType	na
MapNodeType	nd
MapOrientationDirectionEnum	ode

Table 3 (continued)

Data type	Prefix
MapParcelIDType	pid
MapPosition2Dtype	p2
MapPosition3Dtype	p3
MapPriorityType	pr
MapRectangleType	rect
MapReturnType	ret
MapRouteControlType	rc
MapRouteCostModelType	rcm
MapRouteUsageEnum	rue
MapRouteLinkAndCostType	rlc
MapRouteMinimizeOptionEnum	rmoe
MapRoutePointType	rp
MapRoutePointSequenceType	rps
MapRouteDynamicTrafficUsageEnum	rtue
MapSectionIDType	sid
MapSpeedCategoryType	sc
MapSuccessorLinkAndNodeType	sln
MapSuccNodeCostEnum	snce
MapTimeZoneType	tz
MapTollType	tol

Table 4 lists, alphabetically by prefix, the data types corresponding to all Hungarian prefixes used in this International Standard.

Table 4 — Hungarian prefixes used in this International Standard

Prefix	Data type
a	any
afse	AddressFieldStatusEnum
av	MapAccessVehicleType
b	boolean
c	char
ca	MapConditionAttrType
cc	MapConditionCategoryType
ci	MapCursorInfoType
cln	MapConditionLinkAndNodeType
cm	MapConditionModifierType

Table 4 (continued)

Prefix	Data type
cme	MapConditionModifierEnum
cnd	MapConditionType
cod	MapConditionAndDirectionType
coe	MapCursorOriginEnum
crte	MapCursorReturnTypeEnum
cu	MapCursorType
da	MapDateAttrType
dbid	MapDBIDType
dmy	MapDayOfMonthOfYearType
dt	MapDateType
dta	MapAbsoluteDateType
dte	MapDateTypeEnum
dtb	MapDateTimeType
dwe	MapDayOfWeekEnum
dwm	MapDayOfWeekOfMonthType
dwmy	MapDayOfWeekOfMonthOfYearType
dwy	MapDayOfWeekOfYearType
ee	MapEntityEnum
eid	MapEntityIDType
fde	MapFetchDirectionEnum
fte	FeatureTypeEnum
l	long
la	MapLinkAttrType
lc	MapLaneCategoryType
lch	MapLinkCharacteristicsType
lk	MapLinkType
ls	MapLinkSubattrType
na	MapNodeAttrType
nd	MapNodeType
o	octet
ode	MapOrientationDirectionEnum
p2	MapPosition2Dtype
p3	MapPosition3Dtype
pid	MapParcelIDType
pr	MapPriorityType
rc	MapRouteControlType

Table 4 (continued)

Prefix	Data type
rcm	MapRouteCostModelType
rect	MapRectangleType
ret	MapReturnTypes
rue	MapRouteUsageEnum
rlc	MapRouteLinkAndCostType
rmoe	MapRouteMinimizeOptionEnum
rp	MapRoutePointType
rps	MapRoutePointSequenceType
rtue	MapRouteDynamicTrafficUsageEnum
s	short
sc	MapSpeedCategoryType
sid	MapSectionIDType
sln	MapSuccessorLinkAndNodeType
snce	MapSuccNodeCostEnum
tol	MapTollType
tz	MapTimeZoneType
ul	unsigned long
us	unsigned short

The following principles were followed in defining the Hungarian prefixes listed above:

- All (and only) basic data types not containing the word “unsigned” have single-letter prefixes.
- All (and only) basic data types containing the word “unsigned” have prefixes beginning with the letter “u”. The prefixes of these types consist of the letter “u” followed by the prefix of the corresponding signed type.
- All (and only) enumerated types (“enums”) have prefixes ending with the letter “e”.
- Prefixes have been kept as short as possible, consistent with reasonable clarity. No prefix is more than four characters long.

5.3 Application categories

5.3.1 General

The global structure of the API will be specified by using the IDL entities *module* and *interface*.

A *module* is used to group a related set of definitions, e.g. the module "MapRoutePlanning" contains all *interfaces* of the application area Route Planning.

An *interface* describes the set of access functions for accessing a logical data model entity and its relationships to other entities. For example, the interface MapLink contains all access functions that provide the information on a given Link as specified in the requirements drafts, e.g. the requirements starting with "For a given Link ...".

5.3.2 Global module specification

The sub-modules of the API are specified below. Initially, for each application area, a sub-module is specified. In a later stage some modules may be merged depending on the definition of conceptual data sets in the Logical Data Model.

```
module MapAPI {

    module MapRoutePlanning;

    module MapRouteGuidance;

    module MapPositioning;

    module MapDisplay;

    module MapAddressLocation;

    module MapService;

    module MapGeneral;

}
```

5.3.3 Definitions common to all functional categories

5.3.3.1 Constants

5.3.3.1.1 Vehicle access type constants

The following constants are to be used as components of a 32-bit long mask indicating which vehicles are being addressed. The left-most bit is the “tone” designator. If it is set, then the tone of the mask is “all but some vehicle”; if it is not set, then the tone is “some vehicle”. When the “all but” tone is set, the vehicles whose bits are “0” are the ones that are referred to. The second bit from the left is the “all” bit. If it is set, then there is no need to look at individual vehicular bits, it simply states that all entities are selected (or on). Bits 12 through 15 are reserved for additional vehicle types and bits 26 through 29 are reserved for additional control uses in future versions of this International Standard. These 8 bits shall not be used in conjunction with the current version of this International Standard.

```
typedef unsigned long LONG_BITMASK;

typedef LONG_BITMASK V_BITMASK;

const V_BITMASK ALL_BUT_MASK           // 1000 0000 0000 0000 0000 0000 0000 0000
                                        // (0x1 << 31)
                                        = 0x80000000;

const V_BITMASK ALL_ENTITIES_MASK      // 0100 0000 0000 0000 0000 0000 0000 0000
                                        // (0x1 << 30)
                                        = 0x40000000;

const V_BITMASK AUTOMOBILES           // 0000 0000 0000 0000 0000 0000 0000 0001
                                        // (0x1 << 0)
                                        = 0x00000001;

const V_BITMASK ALL_BUT_AUTOMOBILES    // 1011 1111 1111 1111 1111 1111 1111 1110
                                        // (~ALL_ENTITIES_MASK) & (~AUTOMOBILES)
                                        = 0xbfffffff;
```

```
// 0000 0000 0000 0000 0000 0000 0000 0010
// (0x1 << 1)
const V_BITMASK BUSES = 0x00000002;

// 1011 1111 1111 1111 1111 1111 1111 1101
// (~ALL_ENTITIES_MASK) & (~BUSES)
const V_BITMASK ALL_BUT_BUSES = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 0000 0100
// (0x1 << 2)
const V_BITMASK TAXIS = 0x00000004;

// 1011 1111 1111 1111 1111 1111 1111 1011
// (~ALL_ENTITIES_MASK) & (~TAXIS)
const V_BITMASK ALL_BUT_TAXIS = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 0000 1000
// (0x1 << 3)
const V_BITMASK HOVS = 0x00000008; // Carpools

// 1011 1111 1111 1111 1111 1111 1111 0111
// (~ALL_ENTITIES_MASK) & (~HOVS)
const V_BITMASK ALL_BUT_HOVS = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 0001 0000
// (0x1 << 4)
const V_BITMASK PEDESTRIANS = 0x00000010;

// 1011 1111 1111 1111 1111 1111 1110 1111
// (~ALL_ENTITIES_MASK) & (~PEDESTRIANS)
const V_BITMASK ALL_BUT_PEDESTRIANS = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 0010 0000
// (0x1 << 5)
const V_BITMASK BICYCLES = 0x00000020;

// 1011 1111 1111 1111 1111 1111 1101 1111
// (~ALL_ENTITIES_MASK) & (~BICYCLES)
const V_BITMASK ALL_BUT_BICYCLES = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 0100 0000
// (0x1 << 6)
const V_BITMASK TRUCKS = 0x00000040;

// 1011 1111 1111 1111 1111 1111 1011 1111
// (~ALL_ENTITIES_MASK) & (~TRUCKS)
const V_BITMASK ALL_BUT_TRUCKS = 0xbfffffff;

// 0000 0000 0000 0000 0000 0000 1000 0000
// (0x1 << 7)
const V_BITMASK RESIDENT_AND_GUESTS = 0x00000080;

// 1011 1111 1111 1111 1111 1111 0111 1111
// (~ALL_ENTITIES_MASK) & (~RESIDENT_AND_GUESTS);
const V_BITMASK ALL_BUT_RESIDENT_AND_GUESTS = 0xbfffffff;
```

```

// 0000 0000 0000 0000 0000 0001 0000 0000
// (0x1 << 8)
const V_BITMASK DELIVERIES = 0x00000100;

// 1011 1111 1111 1111 1111 1110 1111 1111
// (~ALL_ENTITIES_MASK) & (~DELIVERIES)
const V_BITMASK ALL_BUT_DELIVERIES = 0xbffffeff;

// 0000 0000 0000 0000 0000 0010 0000 0000
// (0x1 << 9)
const V_BITMASK SCHOOL_BUSES = 0x00000200;

// 1011 1111 1111 1111 1111 1101 1111 1111
// (~ALL_ENTITIES_MASK) & (~SCHOOL_BUSES)
const V_BITMASK ALL_BUT_SCHOOL_BUSES = 0xbffffdff;

// 0000 0000 0000 0000 0000 0100 0000 0000
// (0x1 << 10)
const V_BITMASK MOTORCYCLES = 0x00000400;

// 1011 1111 1111 1111 1111 1011 1111 1111
// (~ALL_ENTITIES_MASK) & (~MOTORCYCLES)
const V_BITMASK ALL_BUT_MOTORCYCLES = 0xbffffbfff;

// 0000 0000 0000 0000 0000 1000 0000 0000
// (0x1 << 11)
const V_BITMASK AUTHORIZED_VEHICLES = 0x00000800;

// 1011 1111 1111 1111 1111 0111 1111 1111
// (~ALL_ENTITIES_MASK) & (~AUTHORIZED_VEHICLES)
const V_BITMASK ALL_BUT_AUTHORIZED_VEHICLES = 0xbffff7fff;

// 0000 0000 0000 0001 0000 0000 0000 0000
// (0x1 << 16)
const V_BITMASK Reserved_VEHICLES_01 = 0x00010000;

// 1011 1111 1111 1110 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_01 = 0xbffefffff;

// 0000 0000 0000 0010 0000 0000 0000 0000
// (0x1 << 17)
const V_BITMASK Reserved_VEHICLES_02 = 0x00020000;

// 1011 1111 1111 1101 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_02 = 0xbfffdffff;

// 0000 0000 0000 0100 0000 0000 0000 0000
// (0x1 << 18)
const V_BITMASK Reserved_VEHICLES_03 = 0x00040000;

// 1011 1111 1111 1011 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_03 = 0xbffbffff;

```

```
// 0000 0000 0000 1000 0000 0000 0000 0000
// (0x1 << 19)
const V_BITMASK Reserved_VEHICLES_04 = 0x00080000;

// 1011 1111 1111 0111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_04
= 0xbff7ffff;

// 0000 0000 0001 0000 0000 0000 0000 0000
// (0x1 << 20)
const V_BITMASK Reserved_VEHICLES_05 = 0x00100000;

// 1011 1111 1110 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_05
= 0xbfefffff;

// 0000 0000 0010 0000 0000 0000 0000 0000
// (0x1 << 21)
const V_BITMASK Reserved_VEHICLES_06 = 0x00200000;

// 1011 1111 1101 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~RESERVED_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_06
= 0xbfdfffff;

// 0000 0000 0100 0000 0000 0000 0000 0000
// (0x1 << 22)
const V_BITMASK Reserved_VEHICLES_07 = 0x00400000;

// 1011 1111 1011 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~OPTION_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_07
= 0xbfbfffff;

// 0000 0000 1000 0000 0000 0000 0000 0000
// (0x1 << 23)
const V_BITMASK Reserved_VEHICLES_08 = 0x00800000;

// 1011 1111 0111 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~OPTION_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_08
= 0xbf7fffff;

// 0000 0001 0000 0000 0000 0000 0000 0000
// (0x1 << 24)
const V_BITMASK Reserved_VEHICLES_09 = 0x01000000;

// 1011 1110 1111 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~OPTION_VEHICLES)
const V_BITMASK ALL_BUT_Reserved_VEHICLES_09
= 0xbeffffff;
```

```

// 0000 0010 0000 0000 0000 0000 0000 0000
// (0x1 << 25)
const V_BITMASK Reserved_VEHICLES_10 = 0x02000000;

// 1011 1101 1111 1111 1111 1111 1111 1111
// (~ALL_ENTITIES_MASK) & (~OPTION_VEHICLES)
const V_BITMASK ALL_BUT_ Reserved_VEHICLES_10
    = 0xbdffffff;

// 0011 1111 1111 1111 1111 1111 1110 1111
const V_BITMASK ALL_MOTOR_VEHICLES = 0x3ffffffef; // Without pedestrians

// 1000 0000 0000 0000 0000 0000 0001 0000
const V_BITMASK ALL_BUT_ ALL_MOTOR_VEHICLES // Same as pedestrians (except tone)
    = 0x80000010;

// 0111 1111 1111 1111 1111 1111 1111 1111
const V_BITMASK ALL_ VEHICLE_TYPES = 0x7fffffff; // Includes pedestrians!

// 1100 0000 0000 0000 0000 0000 0000 0000
const V_BITMASK ALL_BUT_ ALL_VEHICLE_TYPES // Same as none (except for tone)
    = 0xc0000000;

typedef V_BITMASK MapAccessVehicleType; // (Hungarian av)

```

5.3.3.1.2 Condition type constants

```

typedef LONG_BITMASK C_BITMASK;

// Prohibited/Allowed single link access or prohibited/mandatory multiple links
// maneuver.
const C_BITMASK TRAFFIC_COND = 0x00000001;

// Information as to whether road is open or closed during construction.
const C_BITMASK CONSTRUCTION_STATUS_COND = 0x00000002;

// Describe monetary cost requirement for travel.
const C_BITMASK TOLL_COND = 0x00000004;

// Travel blocked by physical gate.
const C_BITMASK GATE_COND = 0x00000008;

// Resident and guest regulation.
const C_BITMASK RES_AND_GUEST_COND = 0x00000010;

// This value is defined to be used by the MapGetLinkConditions() function for
// requesting conditions of any type.
const C_BITMASK ANY_COND = 0xffffffff;

typedef C_BITMASK MapConditionCategoryType; // (Hungarian cc)

```

5.3.3.2 Data structures

5.3.3.2.1 LocusType (Hungarian: loc)

This structure defines a position along a link (it can also be just the link).

```
typedef struct locusType_s {  
  
    MapEntityType    link;  
  
    // normalized [0.0, 1.0] fraction  
    // "< 0" means "the entire link" or "unknown",  
    // whichever means something in the context.  
    FractionType     positionAlongLink;  
  
    // [LHS, RHS, both/either, unspecified]  
    SideTypeEnum      side;  
} LocusType;
```

5.3.3.2.2 LocusList

This structure defines a list of loci for use as an origin, destination, or crisp waypoint. The order of elements in this list is not significant.

```
typedef struct locusList_s {  
  
    int                nLoci;  
  
    // list of loci (or set of pointers to loci)  
    LocusType          <sequence>    loci;  
} LocusList;
```

5.3.3.2.3 bbox3PtType

This structure defines a bounding box using 3 points. A bounding box is used to frame requests of data. It is assumed to be a rectangle in some orientation and is described as 3 points (topLeft, bottomLeft, bottomRight). If the 3 points are not corners of the rectangle, the result will be the smallest rectangle such that one edge of the rectangle contains the "bottomLeft" point and the "bottomRight" point and the rectangle contains the "topLeft" point.

```
typedef struct bbox3PtType_s {  
  
    MapPosition2DType    lowerLeftCorner;  
    MapPosition2DType    upperLeftCorner;  
    MapPosition2DType    lowerRightCorner;  
} bbox3PtType;
```

5.3.3.2.4 GeometryTypeEnum

This structure defines geometry types.

```
typedef enum GeometryTypeEnum_e {  
  
    Shape    = -1,  
    Point    =  0,  
    Line     =  1,  
}
```

```

        Area      = 2,
        Complex   = 3
    } GeometryTypeEnum;

```

5.3.3.2.5 FeatureNameType

This structure defines a single name.

```

typedef struct FeatureNameType_s {
    char[3]      charset; // charset of the name Marc code

    string       name;    // the name itself

    short        symbol;  // a symbol associated with the name
                        // like a highway shield
} FeatureNameType;

```

5.3.3.2.6 PointDimensionEnum

If shape points have 2 or 3 dimensions of data.

```

typedef enum pointDimensionEnum_e {

    D2,
    D3
} pointDimensionEnum;

```

5.3.3.2.7 MapPositionType

If shape points have 2 or 3 dimensions of data.

```

typedef struct mapPositionType_s {

    union dimensionType switch (pointDimensionEnum) {
        case D2:      MapPosition2Dtype  point;
        case D3:      MapPosition3Dtype point;
    };
} mapPositionType;

```

5.3.3.2.8 ComponentBoundaryType

This structure defines a single component boundary.

```

typedef struct ComponentBoundaryType_s {

    short        nPts;    // how many points in the boundaries

    MapPositionType <sequence>  point; // the actual list of points

} ComponentBoundaryType;

```